Vol.2011-SE-171 No.1
2011/3/14

メタパターン適用情報に基づく オブジェクトの協調動作履歴可視化ツール

野田訓広[†1]　小林隆志[†1]　阿草清滋[†1]

オブジェクト指向システムの振る舞いの理解支援として，実行履歴からオブジェクトの協調動作の様子を可視化することが有効である．しかし，一般に，実行履歴に含まれる情報量は膨大であるため，可視化される情報を適切に削減・抽象化する必要がある．

本稿では，Pree の提案したメタパターンの適用情報に基づき，関連の強いオブジェクト同士をグループ化することで，オブジェクトの協調動作履歴を抽象化しシーケンス図として可視化するツールを提案する．ツールを複数のオープンソースソフトウェアに対して適用し，その結果を分析することで，提案ツールの有用性を確認した．

A Tool for Visualizing the History of Object Interactions based on Meta Patterns Usage

Kunihiro Noda,[†1] Takashi Kobayashi and Kiyoshi Agusa[†1]

Visualizing object interactions in an execution trace as sequence diagrams is a promising techniques to help developers comprehend the behavior of object-oriented systems effectively. However, in the most cases, the reverse-engineered sequence diagrams contain enormous information, which cause scalability issues.

In this paper, we present a tool for visualizing the history of object interactions which is abstracted by grouping strongly correlated objects based on Pree's meta patterns usage. We applied the tool to open source software and evaluated the effectiveness of our proposed tool in program comprehension tasks.

1. はじめに

ソフトウェア保守作業では保守対象のソフトウェアの構成・振る舞いを十分に理解する必要があり，保守作業の大半は対象ソフトウェアの構成・振る舞いの理解のために費やされる．このため，ソフトウェア理解を目的とした多くの研究が行われている．

オブジェクト指向システムの振る舞いの理解に有効な手法の 1 つとして，実行履歴を UML のシーケンス図を用いて可視化するという方法があるが，一般に，ソフトウェアの実行履歴は膨大なものとなるため，実行履歴から自動生成されたシーケンス図には膨大な量の情報が含まれており，そのままでは読解が困難である．そのため，膨大な情報量を含んだシーケンス図を効果的に探索する手法や，可視化する情報を削減・抽象化し，適切な抽象度でソフトウェアの振る舞いを可視化する手法が必要となる[6]．

本研究の目的は，可視化する情報を削減・抽象化し，オブジェクト指向システムの振る舞いを適切な抽象度で可視化・提示することにより，システムの振る舞いの理解支援を行うことにある．本稿では，Pree の提案したメタパターン[13),14)] の適用情報に基づき，関係の深いオブジェクト同士をグループ化することで，ソフトウェアの振る舞いを抽象化しシーケンス図として可視化するツールを提案する．

関連の強いオブジェクト同士を 1 つのグループとしてまとめ，グループ間の協調動作のみを可視化することで，大局的な振る舞いを効率よく理解することができると考える．さらに，グループ毎の振る舞いを個別に調査することにより，関係の強いオブジェクト群をひとまとまりとして，局所的な振る舞いを効率よく理解できると考える．

提案ツールでは，関連の強いオブジェクト群をグループ化することでオブジェクトの協調動作履歴の抽象化を行う．本研究では，関連の強いオブジェクト群をグループ化する際の基準として，オブジェクト指向システムの設計において多用されるテンプレートとフックの関係に着目する．

テンプレート・フック構造により，システムで共通となる処理 (フローズンスポット) と利用状況に応じて可変となる処理 (ホットスポット) を分離することで，クラス・モジュールの独立性・再利用性を高めることができる．これは，ソフトウェア設計の観点からは非常に有用である．しかし，ソフトウェアの振る舞いを理解するという観点から考えると，テン

†1 名古屋大学 大学院 情報科学研究科
Graduate School of Information Science, Nagoya University

プレート・フック構造により，本来ひとまとまりであるものが複数に分離されていることが，振る舞いの理解を困難にしている要因の 1 つであると考えることもできる．そのため，振る舞いの理解を行う際には，テンプレート・フックとして分離された要素群をひとまとまりとして捉え直し，その役割を理解していくことが有効であると考える．

提案ツールでは，テンプレート・フック構造により結びついているオブジェクト群をグループとしてまとめ，ソフトウェアの振る舞いをグループ間の相互作用として表現する．グループ化による振る舞いの抽象化後には，テンプレート・フック構造により実現される機能が端的かつ簡潔に表現されている状態となるようにグループ化方法を定める．

本研究では，テンプレートとフックの関係に基づくオブジェクトのグループ化方法を具体的に定義するにあたり，Pree の提案したメタパターンに着目する．メタパターンは，テンプレートとフックの関係を，クラスの構成方法の観点から 7 種類に分類したものである．本研究では，メタパターンとして 7 種類に分類されるテンプレート・フック構造の全てに対し，オブジェクトのグループ化方法を具体的に定義する．これにより，あらゆるテンプレート・フック構造に対し，オブジェクトの協調動作の抽象化を図ることが可能となる．

ソフトウェアの実行履歴をシーケンス図を利用して可視化する際，シーケンス図は，実行時間の経過と共に縦軸方向へ大きくなり，登場オブジェクト数の増加と共に横軸方向へ大きくなる．提案ツールによる可視化では，登場オブジェクト数の削減，すなわち，シーケンス図の横軸方向のサイズ縮小が行われることになる．

有用性評価のために，提案ツールを複数のオープンソースソフトウェアに対して適用する．適用結果に対し，可視化される情報量の削減率や，抽象化された協調動作履歴がプログラム理解に有用であるか等を分析することで，提案手法の有用性について議論する．

2. メタパターン適用情報を用いたオブジェクトの協調動作履歴の抽象化

2.1 概　　要

本節では，Pree がメタパターンとして 7 種類に分類したテンプレート・フック構造の全てに対し，オブジェクトのグループ化方法を具体的に定義していく．メタパターンとして分類された 7 種類のテンプレート・フック構造を表すクラス図を**図 1** に示す．

関連の強いオブジェクト群をグループ化しグループ内での相互作用を省略しようとした際，それらのオブジェクト間の相互作用を全て省略してしまうのではなく，それらのオブジェクト群によって実現される機能を端的に表現する振る舞いの様子のみを省略せず残すようにすると，振る舞いの理解により有用であると期待できる．

テンプレート・フック構造では，テンプレートメソッドが処理の大枠を定め，フックメソッドが処理の可変となる部分を実現することになる．ゆえに，テンプレート・フック構造を用いて実現される機能は，テンプレートメソッドの処理内容とフックメソッドのシグネチャのみを提示することで端的に表現できると考える．

そこで，ある 1 つのテンプレート・フック構造により結びついているオブジェクト群をグループ化する際，次の条件を満たすようにグループ化を行うようにする．

- グループ化後もテンプレートメソッドの呼び出しは省略しないようにする．テンプレートメソッドは複数回連鎖的に呼び出されることがあるが，その場合は，1 番初めのテンプレートメソッド呼び出しのみを省略しないようにし，2 番目以降のテンプレートメソッドの呼び出しは省略されるようにする．
- テンプレートメソッドとフックメソッドのシグネチャが異なっている場合は，フックメソッドの呼び出しも省略しないようにする．

本研究では，上記の条件を満たすグループ化の基準として次の関係を用いる．

- ある 1 つのテンプレート・フック構造により結びついているオブジェクト群に属し，“フックメソッド，又は，フックメソッドと同一シグネチャのテンプレートメソッド”を持つオブジェクトをグループ化する．

このグループ化基準によってオブジェクトのグループ化を実施した後，どのグループにも所属しないオブジェクトについては，そのオブジェクトのみを単独の要素として持つグループを作成するようにする．こうすることで，全てのオブジェクトがいずれかのグループに所属

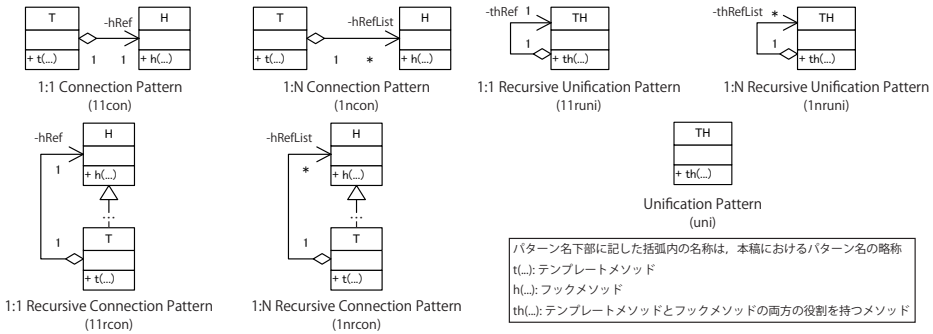

図 1　メタパターンとして分類されたテンプレート・フック構造を表すクラス図
Fig. 1　a class diagram for template-hook structures classified as meta patterns

することとなる．提案ツールでは，オブジェクトのグループ化後，グループ間の協調動作のみを可視化対象することで，システムの大局的な振る舞いを効果的に可視化する．

2.2 オブジェクトのグループ化による協調動作履歴の抽象化

グループ化したオブジェクトの協調動作をシーケンス図上で表現する際には，シーケンス図上において複数のライフラインを 1 つに統合する必要が出てくる．そこで，シーケンス図上におけるグループ間の協調動作の表示方法を次のように定義する．

- オブジェクトグループ G を 1 つのライフラインとして表示する．ライフライン上部の矩形内に表示するオブジェクト名・クラス名としては，G の要素のオブジェクトから任意のものを選びそのオブジェクトのオブジェクト名・クラス名を表示する．このとき，グループ G に名前を付け，オブジェクト名の前に付加する．
- オブジェクトグループ G_i, G_j に対し, オブジェクト $obj_x \in G_i$ がオブジェクト $obj_y \in G_j$ にメッセージを送信するとき，シーケンス図上では次のように表示する．
  - ($i \neq j$ のとき): G_i に対応するライフラインから G_j に対応するライフラインへメッセージが送信されるようにする．
  - ($i = j$ のとき): メッセージを表示しない．

ここで，上記の定義においては，任意の G_i, G_j に対し，$G_i \cap G_i = \phi$ という条件を仮定している．この条件は，2.3.6 節で述べる方法により満たされることになる．

2.3 メタパターン適用部に対するオブジェクトのグループ化

テンプレート・フック構造により実現される振る舞いの種類に応じて 7 種類のメタパターンを 5 通りに分類し，各分類に対しオブジェクトのグループ化方法を 2.3.3 節 から 2.3.5 節で定義する．以降では，テンプレートメソッドを持つオブジェクトをテンプレートオブジェクト，フックメソッドを持つオブジェクトをフックオブジェクトと呼ぶ．

2.3.1 1:1 Recursive Connection Pattern，1:1 Recursive Unification Pattern 適用部に対するオブジェクトのグループ化

ここでは，1:1 Recursive Connection Pattern を例にとってグループ化方法を説明する．1:1 Recursive Unification Pattern でも同様に考えることでグループ化が可能である．

1:1 Recursive Connection Pattern 適用部では，テンプレートオブジェクトがテンプレートオブジェクトまたはフックオブジェクトを (再帰的に) 集約することから，テンプレートオブジェクトとフックオブジェクトの集約のチェーンが形成される．そのため，1:1 Recursive Connection Pattern では，テンプレートメソッド内でテンプレートオブジェクト又はフックオブジェクトに対してメッセージが送信 (テンプレートメソッド又はフックメソッドの呼び出し) されることにより，一連のメッセージ送信のチェーンが形成される．

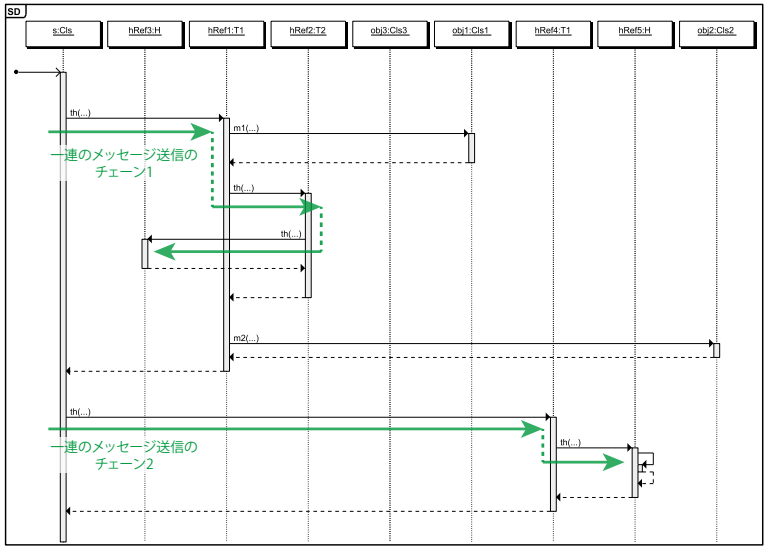

図 2 1:1 Recursive Connection Pattern 適用部の実行時の振る舞いの典型例
Fig. 2 a typical behavior of the part to which 1:1 recursive connection pattern is applied

一般に，1:1 Recursive Connection Pattern では，テンプレートメソッドとフックメソッドのシグネチャは同じもの (本稿ではこれを th() と表記する) となる．そこで，2.1 節で述べたグループ化の基準に従い，集約のチェーンを形成するテンプレートオブジェクトとフックオブジェクトを 1 つのグループとしてグループ化する．

例えば，1:1 Recursive Connection Pattern 適用部の実行時の振る舞いの様子は**図 2** のようになると想定できる．(図 2 において Ti (i は整数) は H のサブタイプである．なお，メッセージ送信チェーンの終端は Ti となる場合もある)．

前述したグループ化方法に従って，図 2 上に登場するオブジェクトのグループ化を行い，2.2 節で述べた方法でグループ間の相互作用のみを可視化すると**図 3** のようになる．ここで，図 3 において，$G1 = \{s\}, G2 = \{hRef_1, hRef_2, hRef_3\}, G3 = \{obj_1\}, G4 = \{hRef_4, hRef_5\}, G5 = \{obj_2\}$ である．

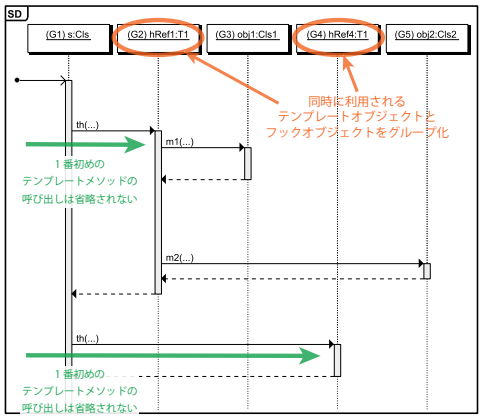

図 3 図 2 の協調動作を抽象化したもの
Fig. 3 the interactions abstracted from the interactions at Fig. 2

2.3.2 1:N Recursive Connection Pattern，1:N Recursive Unification Pattern 適用部に対するオブジェクトのグループ化

ここでは，1:N Recursive Connection Pattern を例にとってグループ化方法を説明する．1:N Recursive Unification Pattern でも同様に考えることでグループ化が可能である．

1:N Recursive Connection Pattern では，テンプレートオブジェクトがテンプレートオブジェクトまたはフックオブジェクトを (再帰的に) 多重に集約することから，テンプレートオブジェクトとフックオブジェクトの集約の関係は木構造を形成することになる．そのため，1:N Recursive Connection Pattern では，テンプレートメソッド内の繰り返し処理でテンプレートオブジェクト群・フックオブジェクト群に対してメッセージが順に送信 (テンプレートメソッド・フックメソッドの呼び出し) されることにより，木構造の各ノードに対してメッセージが伝播していくことになる．

1:N Recursive Connection Pattern では，テンプレートメソッドとフックメソッドのシグネチャは同じものとなる．そこで，2.1 節で述べたグループ化の基準に従い，集約関係の木構造の各ノードにあたるテンプレートオブジェクト・フックオブジェクト群を 1 つのグループとしてグループ化する．

2.3.3 1:1 Connection Pattern 適用部に対するオブジェクトのグループ化

一般に，1:1 Connection Pattern では，テンプレートメソッドとフックメソッドのシグネチャは異なる．テンプレートオブジェクトが集約するフックオブジェクトを切り替えることで，テンプレートメソッドの実行により実現される処理内容を切り替えることができる．

2.1 節で述べたグループ化の基準に従い，1:1 Connection Pattern 適用部に対しては，あるテンプレートオブジェクトのあるテンプレートメソッド t から呼び出されるフックオブジェクト群を 1 つのグループとしてグループ化する．(フックオブジェクト “群” と表記したのは，t が複数回実行されたとき，その実行全てにおいて，t からフックメソッドの呼び出しを受けるフックオブジェクトが常に同一であるとは限らないためである)．

1:1 Connection Pattern では，テンプレートメソッドとフックメソッドは異なるシグネチャを持つため，テンプレートオブジェクトとフックオブジェクトは同一グループとしてグループ化しない．

2.3.4 1:N Connection Pattern 適用部に対するオブジェクトのグループ化

一般に，1:N Connection Pattern では，テンプレートメソッドとフックメソッドのシグネチャは異なり，テンプレートメソッド内の繰り返し処理により複数のフックオブジェクト対してメッセージが送信される．

2.1 節で述べたグループ化の基準に従い，1:N Connection Pattern 適用部に対しては，あるテンプレートオブジェクトのあるテンプレートメソッドから呼び出されるフックオブジェクト群を 1 つのグループとしてグループ化する．1:N Connection Pattern では，テンプレートメソッドとフックメソッドは異なるシグネチャを持つため，テンプレートオブジェクトとフックオブジェクトは同一グループとしてグループ化しない．

2.3.5 Unification Pattern 適用部に対するオブジェクトのグループ化

Unification Pattern ではテンプレートメソッドとフックメソッドのシグネチャは必ず同じものとなる．TH を継承 (実装) したクラスにおいてフックメソッドをオーバーライドすることで，テンプレートメソッドの実行により実現される処理内容を切り替えることができる．

Unification Pattern では，ある 1 つのオブジェクトのみでテンプレート・フックが実現されるため，オブジェクトのグループ化は行わないものとする．

2.3.6 複数のメタパターン同時適用部に対するオブジェクトのグループ化

前述のグループ化方法に従ってグループ化を行った場合，1 つのクラスに対し複数のメタパターンが同時に適用されている箇所では，複数の異なるグループに所属するオブジェクトが出現する可能性がある．そのような場合，シーケンス図上ではライフラインを上手く統合することができず，可視化の際に問題となる．

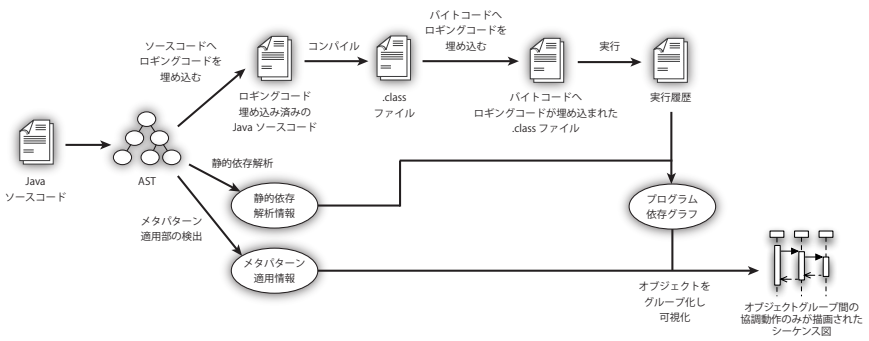

図 4 オブジェクト協調動作履歴可視化ツールの全体像
Fig. 4 an overview of the implemented tool

本研究では，そのような問題へ対処するため，Dugerdil らの用いている方法[8] を参考にする．Dugerdil らは，オブジェクトをグループ化した後，複数のグループに所属するオブジェクトが存在した場合は，複数のグループに所属するオブジェクトのみから構成される新たなグループを作成することで，上述の問題を解決している．本研究でも同様の方法で上述の問題を解決する．複数のグループ $G1, G2, \cdots$ に共通要素が存在している場合，その要素のみからなる新たなグループ G を作成し，$G1, G2, \cdots$ から G の要素を取り除くようにする．

3. メタパターン適用情報を用いたオブジェクト協調動作履歴可視化ツール

本研究では，2 節で述べたオブジェクトのグループ化方法によりオブジェクトの協調動作履歴を抽象化し，シーケンス図として可視化するツールを作成した．ツールの処理対象は Java 言語で実装されたシステムである．ツールの全体像を**図 4** に示す．

本ツールでは，ユーザ記述コード部分のみを処理対象とし，オブジェクトのグループ化，及び，グループ間協調動作の可視化を行う．ライブラリ内部の処理内容に関しては可視化を行わない．以降，簡単に実装詳細を述べる．

■メタパターン適用部の検出　ソースコードを AST に変換した後，AST 全体を 2 回探索することで，メタパターン適用部を検出する．

(1)　全てのメソッド定義 (AST における MethodDeclaration ノード)，メソッド呼び出し (AST における MethodInvocation, SuperMethodInvocation ノード) を探索し，オーバーライドしている・されているメソッドを検出する．(フックメソッドを検出する)．

(2)　全てのメソッド呼び出し式 (AST における MethodInvocation, SuperMethodInvocation ノード) を探索し，上で検出したフックメソッドを呼び出しているメソッドを検出する．(テンプレートメソッドを検出する)．同時に，テンプレートメソッドを持つクラスがフックメソッドを持つクラスを多重度いくつで集約しているかを調べる．

(3)　以上の情報を総合し，どのメタパターンが適用されているかを判定する．

上記の処理の内，集約の多重度の調査については，配列型の変数を介して集約している場合は「0..*」，配列型でない変数を介して集約している場合は「0..1」としている．また，標準ライブラリを介した頻出する集約関係の考慮として，「java.lang.Iterable を実装する型の変数を介して集約しており，かつ，Iterable の総称型としてフックメソッドを持つクラスの型が指定されている」という場合は多重度「0..*」で集約しているものとする．

今回は集約と使用 (クラス図における関連名 use で繋がれる関係，ローカル変数による結びつき) の関係を区別せずにメタパターンの検出を行う．すなわち，ローカル変数を介してフッククラスのオブジェクトのフックメソッドが呼び出される場合にも，擬似的にメタパターンが適用されているものとして扱う．

ここで，上述した集約の判定方法 (Iterable を介した集約判定方法) はある種のヒューリスティクスを用いており，集約の多重度が正確に判定できない可能性があることに注意する．しかし，多重度が判定できなくとも，2 節で述べたオブジェクトのグループ化を実行する際に影響はない．多重度が判定不能の場合は，多重度「0..*」で集約していると仮定してグループ化を行えば，多重度が完全に判定できている場合と同等の結果を得ることができる．

■静的依存解析　静的依存解析部分に関しては，我々の先行研究[11] において作成したシーケンス図スライシングツール Reticella を拡張する形で実装している．AST の情報を利用することで，データ依存関係・制御依存関係を解析し，依存グラフの雛形を作成する．「雛形」と表記したのは，実行時情報を併合することで，完全なプログラム依存グラフとなるためである．なお，2 節で述べたグループ化を実行する際にはデータ依存関係や制御依存関係は必要ないが，これは，今後の研究を考慮してツールを作成したためである．

プログラム依存グラフは先行研究[11] において定義した B-Model に基づくデータ形式で作成される．B-Model はプログラムの振る舞いを表現するモデルであり，B-Model を利用することで，実行履歴とシーケンス図の相互変換を容易に行うことができる．

■ロギングコードの埋め込み　実行履歴を取得するために，ロギングコードの埋め込み

表 1 実験対象のオープンソースソフトウェア
Table 1 target software of the experiment

名称	バージョン	規模	
		ソースファイル数	LOC (コメント・空行含む)
jpacman[4]	rev.53	40	5957
JHotDraw[3]	5.3	195	27712
Checkstyle[1]	5.3	295	49022

を実施している．AST の情報を基にソースコードへロギングコードを埋め込んだ後，ロギングコード埋め込み済みのソースコードをコンパイルして.class ファイルを生成する．その後，.class ファイル内のバイトコードに対して，バイトコード変更ライブラリ Javassist[2] を用いて，更にロギングコードを埋め込む．ロギングコードの埋め込みを 2 回行っている理由は，どちらか片方だけでは得られる情報量が不十分となるためである．

ロギングコード埋め込みにより，Java 言語における変数定義・参照，条件分岐・繰り返し処理の開始・終了，メソッドの呼び出し・復帰，フィールドアクセス等をイベントとして検出・記録できるようにしている．

■オブジェクトグループ間の協調動作の可視化 提案手法に基づきオブジェクトをグループ化した後，オブジェクトグループ間の協調動作をシーケンス図として可視化する．可視化の際には，Quick Sequence Diagram Editor[5] というシーケンス図描画アプリケーションを利用している．

4. 評価実験

4.1 概　　要

提案ツールの有用性を評価するために，3 つのオープンソースソフトウェアへの適用実験を行った．適用実験では，オブジェクトのグループ化によりどのようなグループが作成されるのかを調査した．また，抽象化された協調動作履歴がプログラム理解にとって有用となるかをツールの出力であるシーケンス図を分析することで評価した．

4.2 実験方法

実験対象の 3 つのオープンソースソフトウェアを**表 1** に示す． 本実験では，以下の実行シナリオに沿って 3 つのソフトウェアを動作させ，その実行履歴に対し提案ツールを適用する．実行シナリオはどれも各ソフトウェアの代表的な機能を実行するものである．

- jpacman: プログラムを起動しゲームを開始．ゲーム開始後，パワークッキーを取得し，ゴーストと接触しゴーストを食べる．その後，プログラムを終了する．
- JHotDraw: 付属のドローアプリケーション (CH.ifa.draw.samples.javadraw.JavaDrawApp) を実行．ウィンドウサイズを大きくした後，矩形を 2 つ描きプログラムを終了．
- Checkstyle: 付属の Sun Coding Convensions を用いて，Checkstyle の Main クラスのソースコード (Main.java) がコーディング規約に従っているかを検査する．

表 2 ソフトウェア実行時にロードされたクラス数，及び，実行履歴に含まれる情報量
Table 2 the number of loaded classes, messages and objects on runtime

ソフトウェア名	ロードクラス数	発生総イベント数	発生メッセージ数	登場オブジェクト数
jpacman	69	10441153	3922814	3505
JHotDraw	152	1079463	307727	24469
Checkstyle	214	1284171	380099	16454

表 3 各ソフトウェアのメタパターン適用箇所数
Table 3 meta patterns in the target software

ソフトウェア名	uni	11con	1ncon	11rcon	1nrcon	11runi	1nruni
jpacman	0	29	92	0	14	0	0
JHotDraw	334	172	938	87	119	0	1
Checkstyle	106	46	570	2	21	1	0

図 1 で示したパターン名の略称と対応させて記述している．検出箇所数はテンプレートメソッドとフックメソッドの組の数である．検出箇所数は，フックメソッドがライブラリメソッドであるものも含む．テンプレートメソッドがライブラリメソッドであるものは含まない．

4.3 結　　果

■ソフトウェア実行の結果得られた情報量 実行時に Java VM にロードされたクラス数，及び，得られた実行履歴に含まれる情報量を**表 2** に示す． シーケンス図の描画対象となるのは，メッセージ送信，及び，オブジェクトである．表 2 における発生メッセージ数，登場オブジェクト数がシーケンス図の描画時の大きさに影響を与える．

■メタパターン適用箇所の検出結果 各ソフトウェアに対してメタパターン適用箇所を検出した結果を**表 3** に示す．

前述した通り，本ツールでは，集約の多重度が正確に判定できない箇所がある．そのような箇所については，多重度「0..*」で集約しているとして適用箇所数を集計した．また，本ツールでは，メタパターン検出にあたり，クラス間の集約関係と使用の関係 (ローカル変数による結びつき) を区別していない．そのため，表 3 に示した結果は，他研究における検出数に比べ，検出箇所数が多くなっている．

表 4 オブジェクトのグループ化の結果
Table 4 the result of grouping objects

ソフトウェア名	総グループ数	単一要素グループ数	複数要素グループ数
jpacman	2461 (3505)	2444 (2444)	17 (1061)
JHotDraw	22607 (24469)	21192 (21192)	1415 (3277)
Checkstyle	15927 (16454)	15870 (15870)	57 (584)

各項目の括弧内の値は，グループに含まれるオブジェクトの総数

■オブジェクトのグループ化結果 オブジェクトのグループ化の結果を表 2 に示す．また，グループ間の協調動作を表現した出力シーケンス図の一例を**図 5** に示す．図 5 は jpacman に対する適用結果の一部である．

4.4 考　　察

メタパターンの検出数について，表 1 に示した総ファイル数・LOC と，表 3 に示した結果を比較することで，今回実験対象とした 3 つのソフトウェアには多くのメタパターン適用箇所が存在することが分かるが，これは今回実験対象としたソフトウェアに限ったことではないと考える．テンプレート・フック構造はオブジェクト指向プログラミングにおいてごく一般的に利用されるものであるため，多くのオブジェクト指向システムにおいて，多くのメタパターン適用箇所が存在すると期待できる．そのため，多くのオブジェクト指向システムに対して，提案手法の情報量削減効果が期待できる．

次に jpacman の適用結果を取り上げて，提案手法によるグループ化の有用性を考察する．jpacman では，パックマン・ゴースト・クッキーの衝突判定・処理において，GoF の Observer パターンを用いた衝突の通知が行われており，その部分にテンプレート・フック構造が使われているためグループ化の対象となった．パックマン・ゴースト・クッキー等の描画対象オブジェクト群がグループ化されることで，描画される側と描画する側という 2 大要素の大局的な相互作用としてシステムの振る舞いが可視化されるようになった．これは，システムの大局的な振る舞いを効果的に可視化するという提案手法の目的に合致しており，振る舞いを理解する上で有用であると考える．

また，パックマン・ゴーストの状態 (パワー状態・ブリンク状態・死状態など) を表現するために State パターンが利用されており，状態を表すオブジェクト群がグループ化の対象となった．このグループ化により，パックマンが複数のオブジェクトを用いて状態遷移を行っているという情報は隠蔽されるようになった．システム全体の大局的な振る舞いから考えると，パックマンの状態遷移というのは局所的なものであると考えることができるため，このグループ化により，システムの大局的な振る舞いが効果的に表現されるようになったと考える．

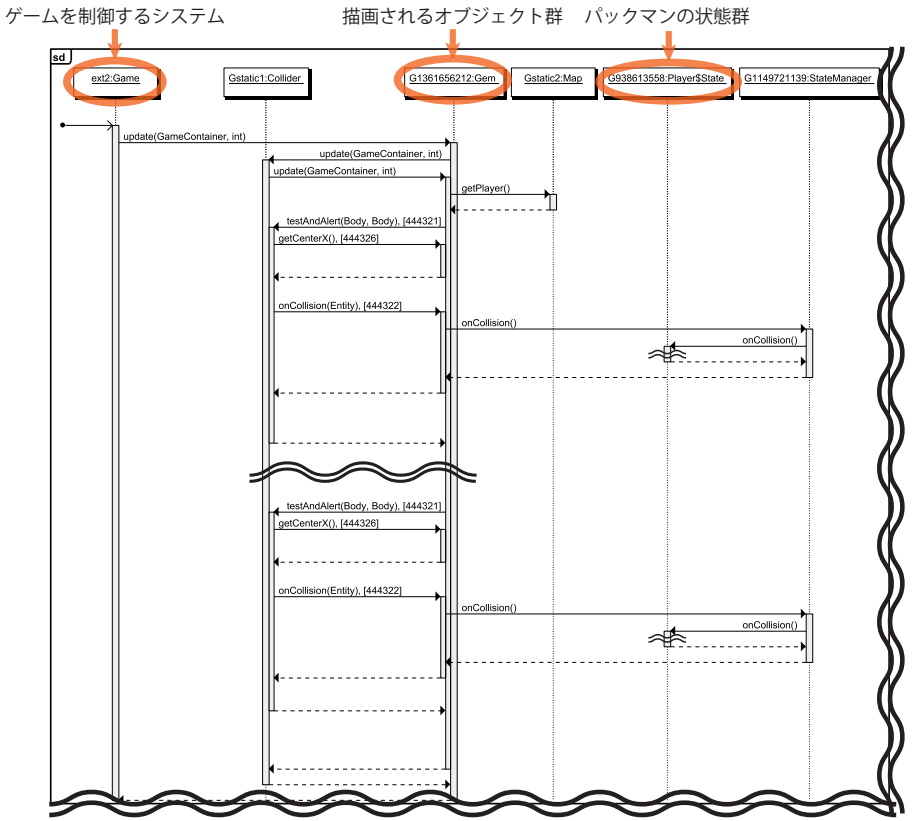

図 5 出力シーケンス図の一部 (jpacman に対する適用結果の一部)
Fig. 5 a part of the sequence diagram resulted from the application to jpacman

このようなグループ化の結果図 5 に示すようなシーケンス図が出力された．図 5 では，描画する側・描画される側・パックマンの状態といったグループ間の相互作用としてシステムの振る舞いが可視化されている．ゲームを制御するシステム側から update() メソッドによる更新通知を受けた後，描画されるオブジェクト群がユーティリティ Collider クラスの

メソッドを用いて，衝突判定・衝突処理 (testAndAlert, onCollision メソッド) を行っている．衝突していると判定された場合には，パックマンの状態に応じた振る舞いを行うために，パックマンの状態オブジェクト群に対しメッセージを送信 (onCollision メソッド) している様子が把握できる．図 5 は大局的な振る舞いを端的に表現しており，システムの振る舞いを理解する上で有用な情報となることが期待できる．

また，グループ化の結果，単一要素グループが多く作成されていることが分かる．単一要素のグループが多く存在することから，提案手法によるグループ化の効果が低いように思えるが，このことは，以下に述べる理由から，提案手法の有用性を損なうものではないと考える．

詳細を調査した結果，単一要素グループに含まれるオブジェクトの大半 (7 割から 8 割以上) は，一時的にしか利用されないオブジェクトや振る舞いを理解する必要性が低いオブジェクトであった．

例えば, jpacman の場合では, 一度生成されたらそれ以降ほぼ利用されることがない Block, Rectangle, Vector, Date クラスのオブジェクトが 2400 個程度，単一要素グループに含まれるオブジェクトとして存在していた．Block クラスのオブジェクトはゲーム開始時に一度生成された後は，何もメッセージを送受信せず，Rectangle クラスのオブジェクトは Block クラスのオブジェクトの大きさ等を定義するためだけに利用されていた．JHotDraw の場合では，Rectangle2D$Double, QuadTree, Hashtable, Vector クラスのオブジェクトが 18000 個程度，単一要素グループに含まれるオブジェクトとして存在していた．QuadTree クラスのオブジェクトは，指定された領域内にオブジェクトが存在しているか否かを判定するために作成され，Rectangle2D$Double クラスのオブジェクトは，QuadTree オブジェクトに関する領域を指定するために作成されていた．オブジェクトの存在判定は再帰的に行われ，その度に新たなオブジェクトが生成されるため，多くのオブジェクトが生成されることになるが，本質的には，ある 1 つの QuadTree オブジェクトについてのみ振る舞いを理解すればよい．また，Hashtable, Vector といったクラスのオブジェクトはユーティリティとして利用されているだけであり，JHotDraw というソフトウェアの振る舞いを理解する上でそれほど重要ではない．

このように，単一要素グループに含まれるオブジェクトの大半は，一時的にしか利用されないものや，振る舞いを詳細に理解する必要の低いオブジェクトであった．これは，一時的にしか活性化しないオブジェクトは，テンプレート・フックとして振る舞うことがないためグループ化の対象とならないと想定できることからも正しいと言える．

以上の考察から，単一要素グループに含まれるオブジェクトに関しては，クラス単位でオブジェクトをグループ化する等の簡単な方法により別途グループ化しても問題ないと想定でき，その場合，単一要素グループに含まれるオブジェクトは少数のグループにまとめることが可能である．システムの振る舞いを理解するうえで重要なオブジェクトのグループ化 (jpacman における State クラスのオブジェクトのグループ化など) は確実に行われているため，上述したような対処を行えば，提案手法の有用性は損なわれないと考える．

4.5 評価の妥当性

Pree の定めたメタパターンは，構造に関する関係のみを規定しており，各オブジェクトの役割を具体的に規定しているわけではない．そのため，今回の提案手法は広い適用範囲を持つことになる．しかし，今回グループ化の対象としたオブジェクト群には，ある程度の関連の強さが存在していると期待できる一方で，関係の弱いオブジェクト同士が 1 つのグループにまとめられてしまう危険性もある．

この問題については，今後，より具体的なデザインパターン (例えば GoF のデザインパターン) に基づくグループ化方法を検討していくことで解決できると考える．より具体的なデザインパターンに基づいてグループ化方法を検討すると，各オブジェクトの役割がより明確なものとなるので，関連の弱いオブジェクト群をグループ化してしまう危険を低減できる．ただし，抽象化の程度は小さくなるので，膨大な情報量を適切に扱えるようによりよいグループ化の手段を議論する必要がある．

多くのデザインパターンは，提案手法で着目したメタパターンを複数組み合わせた上で，各オブジェクトの役割を規定したものであると捉えることができるため，提案手法のグループ化方法を自然な形で，より具体的なパターンに基づくグループ方法へと発展させていくことが可能だと考える．

また，関連は強いがグループとしてまとめられることが適切ではないオブジェクト群が存在する場合もある．例えば，4.4 節で挙げた，パックマン・ゴースト・クッキー等のオブジェクトがグループ化されることについて考える．確かに，描画する側と描画される側という 2 大要素の相互作用として振る舞いが可視化されるようにはなるが，パックマン・ゴースト・クッキーのそれぞれは明らかに担う役割が異なっており，それらがすべて 1 つにまとめられることが常に有用であるとは断言できない．今後，オブジェクトの担う役割をテンプレート・フック構造以外の観点からも捉えるようにし，より有用なグループ化方法を検討していく必要がある．

今回は 3 つのオープンソースソフトウェアに対して適用実験を行い手法の有用性を議論

したが，他のソフトウェアに対しても同様の議論が可能であるとは断定できない．しかし，今回情報量削減のために着目したテンプレート・フック構造は，オブジェクト指向プログラミングにおいてごく一般的に利用される構造であり，多くのオブジェクト指向システムにおいて多数含まれていることが期待できることから，あらゆるテンプレート・フック構造に対して適用可能な提案手法のオブジェクトグループ化方法は，多くのオブジェクト指向システムに対して有効であると考える．

今後，提案手法の有用性の程度をより明確にするため，更なる評価実験が必要である．より規模の大きなソフトウェアへの適用を実施したり，被験者実験等で有用性をより客観的に評価することが必要であると考える．

5. 関 連 研 究

5.1 オブジェクト協調動作履歴の抽象化による実行履歴の情報量削減手法

Taniguchi らは，プログラム実行時の繰り返し処理や再帰呼び出しなどにより生成される，類似した振る舞いを圧縮する手法を提案している[16]．Hamou-Lhadj らは，実行履歴から，utility 等の実装詳細 (システムの主要なコンセプトを実装しているわけではない要素) を取り除くことにより，実行履歴の要約 (summary) を取得する手法を提案している[9]．伊藤らは，実行履歴におけるメソッド呼び出し関係グラフに対し支配関係を計算し，支配関係に基づいたオブジェクトのグループ化方法を定義している[19]．Dugerdil らは，ある単位時間内に発生するオブジェクト間の相互作用の回数に着目し，関連の強いオブジェクト群をクラスタとしてまとめ，クラスタ間の相互作用をシーケンス図として表現することで，実行履歴の情報量の大幅な削減を図っている[8]．

5.2 着目した特定の振る舞いのみを実行履歴から抽出する手法

宗像は，オブジェクト群の同値分割を行う手法を提案している[18]．着目クラスのオブジェクトの振る舞いの類似性に基づき，オブジェクト群の同値分割を行うことで，クラスの代表的な動作シナリオを効果的に可視化できるようにしている．我々もこれまでに，プログラムスライシングを応用することにより，システムの振る舞い全体の中から，着目している箇所とそれに関連する部分のみをシーケンス図として可視化する手法を開発した[11]．Watanabe らは，オブジェクトの生成タイミングに着目して，実行履歴を複数の “phase” に分割する手法を提案しており[17]，その手法は，Ishio らの作成した実行履歴からのシーケンス図生成ツール AMIDA[10] 上で実現されている．現在活性化しているオブジェクトの集合の変化を観測することで，タスクの開始地点 (すなわち，“phase” の開始地点) を特定することで，実行履歴を複数の “phase” に分割する．

5.3 実行履歴を効果的に可視化・探索する手法

Sharp らは，シーケンス図に対する効果的な探索方法を複数提案している[15]．単純にシーケンス図に対する拡大・縮小を可視化ツールの機能として提供してもあまり大きな有用性は得られず，時間やコールスタックの深さに基づくフィルタリングなどの情報量削減手法と併用する必要があるなどの点について主張している．

Cornelissen らは，ソフトウェアの構造情報を円形に表示するビューと実行時の振る舞いを時系列に沿って表示するビューを同時に提示することで，ソフトウェア理解支援を図っており，限られた表示領域内に効果的に情報を表示する工夫がなされている[7]．

5.4 提案手法との比較

提案手法は，5.1 節で述べた手法と同種の手法であり，それらの手法は，システム全体の振る舞いを効率よく理解したいときに有用である．5.2 節で述べた手法は，着目しているある特定の要素に関する部分のみを効率よく理解したいときに有用である．5.3 節で述べた手法は，「膨大な情報量を含む実行履歴をどう可視化すれば効果的か」という可視化の部分に重点を置いており，その点において提案手法や 5.1 節・5.2 節で述べた手法と異なっている．以降では，5.1 節で述べた手法と提案手法との比較について述べる．

提案手法では，オブジェクトのグループ化によりオブジェクトの協調動作履歴の抽象化を行う．そのため，実行履歴において，主にオブジェクト数が大幅に削減され，振る舞いをシーケンス図として可視化した際には，横軸方向へシーケンス図が圧縮されることになる．それに対し，Taniguchi らの手法[16] や Hamou-Lhadj らの手法[9] では，実行履歴において，主にメッセージ数の大幅な削減を行うため，振る舞いをシーケンス図として可視化した際には，縦軸方向へシーケンス図が圧縮されることになり，圧縮方向が提案手法と異なっている．伊藤らの手法[12] や Dugerdil らの手法[8] は，オブジェクトのグループ化による実行履歴の情報量削減を行っており，その点において提案手法と類似している．

「伊藤らの手法」と，「Dugerdil らの手法，及び，提案手法」との違いは，オブジェクトのグループ化の際に，オブジェクト間の関連の強さを考慮しているか否かという点にある．伊藤らは，メソッド呼び出し関係グラフの “構造” に着目してグループ化可能なオブジェクト群を特定している．それに対して，Dugerdil らの手法，及び，提案手法では，何らかの方法で “関係の強い” オブジェクト群を特定することで，オブジェクト間の関連の意味を考慮したグループ化を行っている．

Dugerdil らの手法と提案手法との違いは，オブジェクト間の関連の強さの捉え方にある．

Dugerdil らの手法は，単位時間内の相互作用回数に着目しているため，関連のない 2 つ以上の異なる処理がソースコード上の近い位置に混在して記述されている場合，関連のないオブジェクト同士が 1 つのグループにまとめられてしまう可能性があるが，抽象化の度合いは大きい．それに対し提案手法は，テンプレート・フック構造，すなわち，設計情報に着目し関連の強いオブジェクト群を特定するため，関連のないオブジェクト同士が 1 つのグループにまとめられてしまう恐れは少ないが，その分抽象化の度合いは小さくなる．

システムの振る舞い理解の際には，まず，比較的抽象度の高い手法でシステムの全体の振る舞いの概要を理解した後，抽象度の低い手法でより詳細な振る舞いを理解していくという top-down のアプローチ方法が有効な手段の 1 つになると考える．

6. おわりに

6.1 まとめ

本稿では，Pree の提案したメタパターンの適用情報に基づき，関係の深いオブジェクト同士をグループ化することで，ソフトウェアの振る舞いを抽象化し可視化するツールを提案した．提案ツール複数のオープンソースソフトウェアに適用し，その適用結果から，提案ツールのプログラム理解に対する有用性を確認した．

6.2 今後の課題

提案手法のより詳細な有用性評価

より詳細な有用性評価を行うため，より規模の大きなソフトウェアへの適用を実施したり，被験者実験を行うことでプログラム理解への有用度をより客観的に評価する必要がある．

より具体的なデザインパターン適用情報を考慮した協調動作履歴の抽象化

提案手法では，メタパターン適用情報を用いて，オブジェクトのグループ化を行った．今後は，より具体的なパターン (例えば GoF のデザインパターン) の適用情報を利用した，より有用な協調動作履歴の抽象化手法の提案を検討する．より具体的なパターンの適用情報を考えることで，オブジェクトの役割がより明確となる．そのため，オブジェクトのグループ化方法をより適切に定義でき，プログラム理解にとってより有用になると考える．

謝辞 本研究の一部は科研費 (20300009, 21700027, 22240005) の助成により行われた．

参考文献

1) : Checkstyle, http://checkstyle.sourceforge.net/.
2) : Javassist, http://www.csg.is.titech.ac.jp/~chiba/javassist/.
3) : JHotDraw, http://www.jhotdraw.org/.
4) : jpacman, http://code.google.com/p/jpacman/.
5) : Quick Sequence Diagram Editor, http://sdedit.sourceforge.net/.
6) Bennett, C., Myers, D., Storey, M.-A., German, D.M., Ouellet, D., Salois, M. and Charland, P.: A survey and evaluation of tool features for understanding reverse-engineered sequence diagrams, *J. Softw. Maint. Evol.*, Vol. 20, No. 4, pp. 291–315 (2008).
7) Cornelissen, B., Zaidman, A., Holten, D., Moonen, L., van Deursen, A. and van Wijk, J.J.: Execution trace analysis through massive sequence and circular bundle views, *J. Syst. Softw.*, Vol.81, No.12, pp.2252–2268 (2008).
8) Dugerdil, P. and Repond, J.: Automatic Generation of Abstract Views for Legacy Software Comprehension, *ISEC '10: Proceedings of the 3rd India software engineering conference*, pp.23–32 (2010).
9) Hamou-Lhadj, A. and Lethbridge, T.: Summarizing the Content of Large Traces to Facilitate the Understanding of the Behaviour of a Software System, *ICPC '06*, pp.181–190 (2006).
10) Ishio, T., Watanabe, Y. and Inoue, K.: AMIDA: a Sequence Diagram Extraction Toolkit Supporting Automatic Phase Detection, *Proc. ICSE Companion '08*, pp. 969–970.
11) Noda, K., Kobayashi, T., Agusa, K. and Yamamoto, S.: Sequence Diagram Slicing, *APSEC '09*, pp.291–298 (2009).
12) Posnett, D., Bird, C. and Devanbu, P.: THEX: Mining Metapatterns from Java, *MSR 2010*, IEEE, pp.122–125 (2010).
13) Pree, W.: Meta Patterns - A Means For Capturing the Essentials of Reusable Object-Oriented Design, *ECOOP '94*, pp.150–162 (1994).
14) Pree, W.: *Design Patterns for Object-Oriented Software Development*, ACM Press/Addison-Wesley Publishing Co. (1995).
15) Sharp, R. and Rountev, A.: Interactive Exploration of UML Sequence Diagrams, *VISSOFT '05*, pp.1–6.
16) Taniguchi, K., Ishio, T., Kamiya, T., Kusumoto, S. and Inoue, K.: Extracting Sequence Diagram from Execution Trace of Java Program, *IWPSE '05*, pp.148–154 (2005).
17) Watanabe, Y., Ishio, T. and Inoue, K.: Feature-level Phase Detection for Execution Trace Using Object Cache, *WODA '08*, pp.8–14 (2008).
18) 宗像聡：プログラム動作理解支援を目的としたオブジェクトの振舞いの同値分割手法，修士論文，大阪大学 (2010).
19) 伊藤芳朗，渡邊結，石尾隆，井上克郎：オブジェクトの動的支配関係解析を用いたシーケンス図の縮約手法の提案，情報処理学会研究報告, Vol.2008, No.55，pp.9–16 (2008).